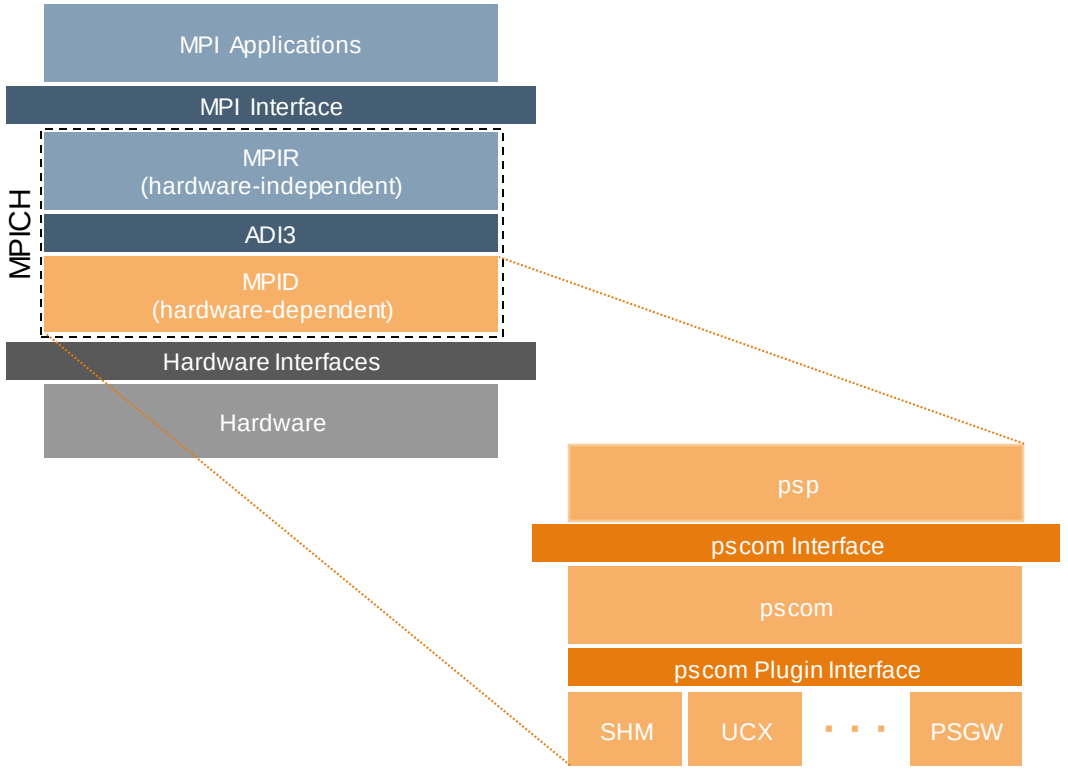
Malleability with MPI Sessions
Extensions for ParaStation MPI

Sonja Happ, ParTec AG

6 May 2024

ParaStation MPI

- Based on MPICH 4.1.2
- Supports MPICH tools for tracing, debugging, etc.
- Integrates into MPICH on the MPID layer by implementing an ADI3 device
- The PSP Device is powered by pscom: A low-level point-to-point communication library
- Support for various transports/ protocols via pscom plugins (InfiniBand, Omni-Path, BXI, etc.)
- Concurrent usage of different transports
- Repositories
  - https://github.com/ParaStation/psmpi
  - https://github.com/ParaStation/pscom

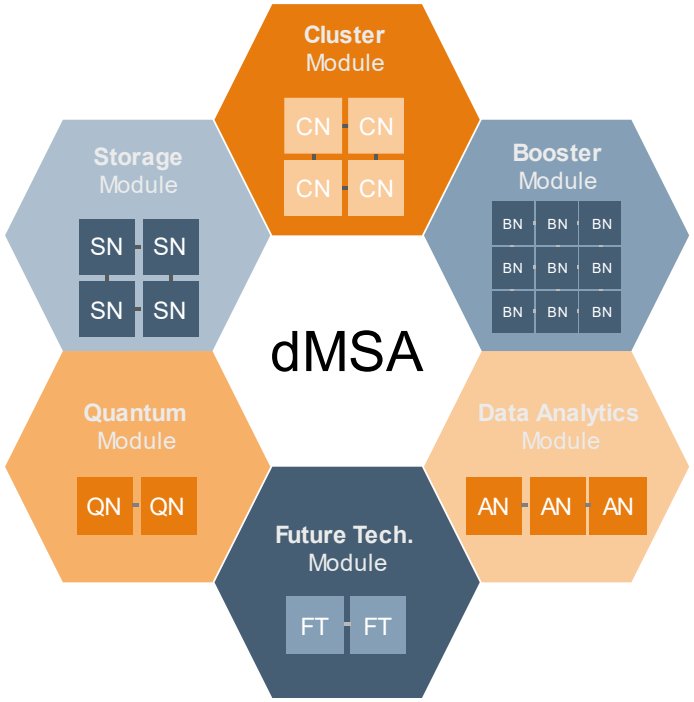


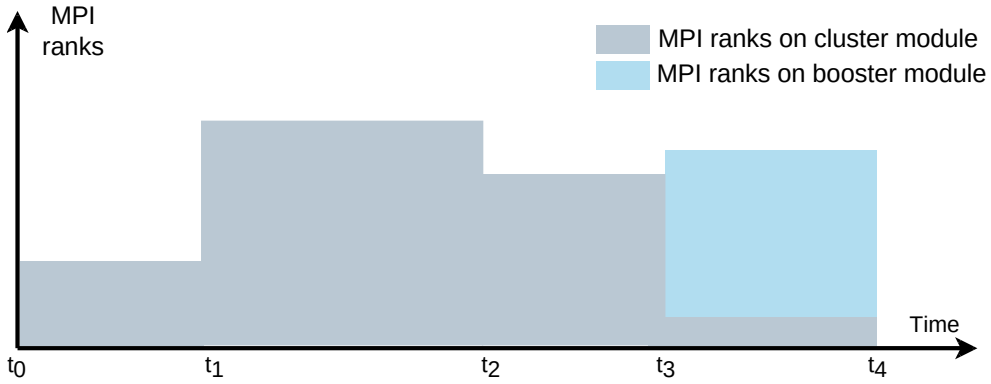
ParaStation MPI

Malleability – Dynamic resource changes for MPI applications

Dynamic change of MPI ranks

- Switch between computational phases
- Exploit dynamic Modular System Architecture (dMSA)
- Fulfill external constraints





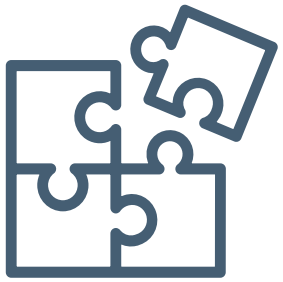
Potential scenario:
t_0: Launch app with MPI ranks on cluster module
t_1: Expand app on cluster module (e.g. new app phase)
t_2: Shrink app on cluster module (e.g. energy constraint)
t_3: Shrink app on cluster module, start ranks on booster module (e.g. new app phase)
t_4: Finish app

Challenges for MPI libraries

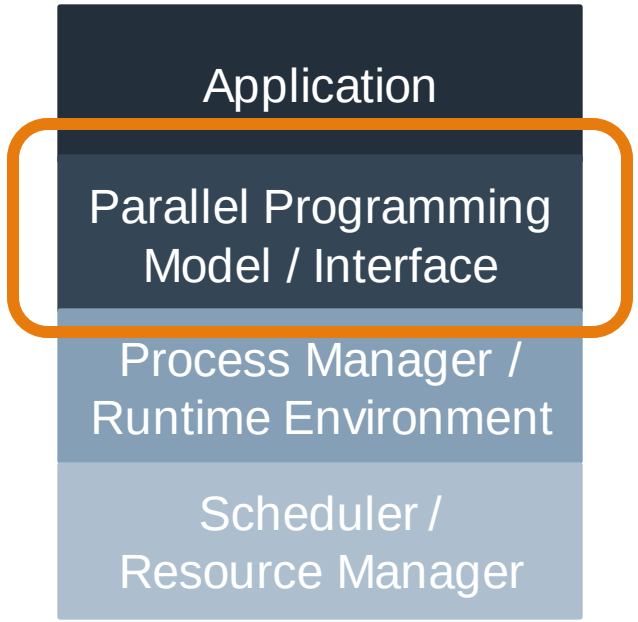
- MPI interface for malleability
- Exchange with process manager/ runtime environment
- Management of resource changes at runtime

Software stack for computational malleability in HPC

- Enhancements required in all layers of SW stack
  - Programming models/ interfaces such as MPI are one piece of the puzzle
  - Apps, runtime environments, and scheduling need to do their parts



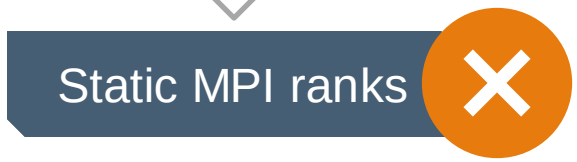
- Other relevant areas
  - Monitoring & profiling
  - I/O & storage

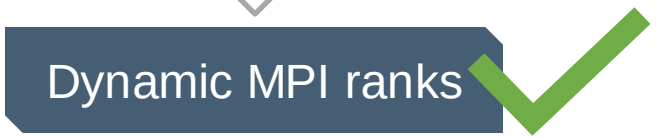


MPI Sessions for malleability

MPI World
No re-initialization of MPI library, `MPI_COMM_WORLD` used to derive groups and communicators

MPI Session
Re-initialization of MPI library via consecutive MPI Sessions, `MPI_COMM_WORLD` not available, use process sets to derive groups and communicators





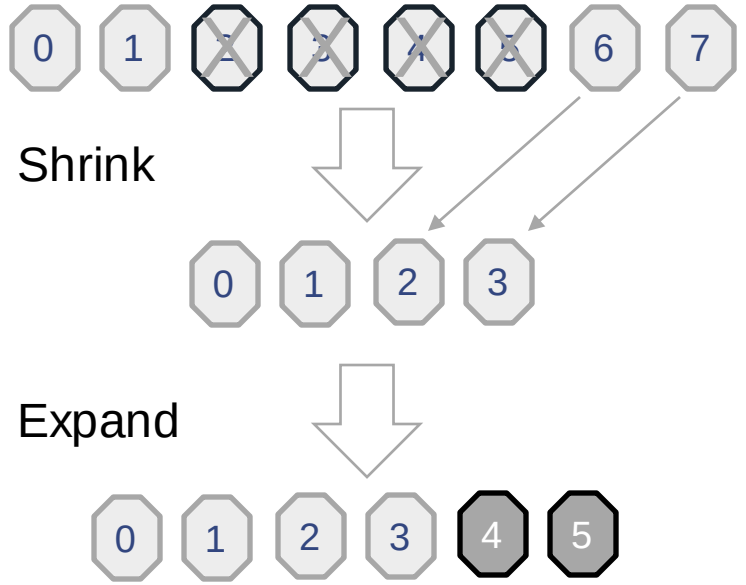
- Exploit re-initialization ability of MPI Sessions
  - Re-initialize MPI library for changed resources
  - Update process sets during re-initialization to reflect changed MPI ranks

- Requirements for malleable MPI applications
  - Must use MPI Session model and process sets
  - Must not use MPI World model and `MPI_COMM_WORLD`

What to consider?

MPI Library

- Global MPI rank numbering without gaps or overlaps
- Dynamic processes with MPI Sessions
  - Overcome limitations of `MPI_Comm_spawn`
  - Asynchronous resource management
- Make use of PMIx concepts & interfaces



MPI Users

- Think in ranks and not nodes
- Select starting point of malleability operation
- Reuse existing utility functions for MPI Sessions

> *"The MPI procedures described in this section require the World Model, meaning that MPI_INIT or MPI_INIT_THREAD has been used to initialize MPI."*
>
> MPI 4.1, Section 11.7 "The Dynamic Process Model", end of 1. paragraph

Proposed MPI eXtensions for MPI Session re-initialization

Re-initialization of MPI library – called by all processes:

```
int MPIX_Session_reinit(
   int count                                          /*Number of MPI sessions*/,
   MPI_Session **session_array_in                     /*Array of all MPI Sessions*/ ,
   MPI_Session *session_out                           /*Newly initialized MPI Session - output*/,
   MPI_Info info                                      /*Info object for init of session_out*/,
   MPIX_Session_reinit_preparation_function *prep_fn  /*Callback to prepare session finalization*/ ,
   void *prep_userdata                                /*App-specific data to use in prep_fn*/,
   MPIX_Session_reinit_resume_function *resume_fn     /*Callback to resume after re-init*/,
   void *resume_userdata                              /*App-specific data to use in resume_fn*/ ,
   int leave                                          /*1: Process terminates, 0: Process stays*/,
   MPI_Count nleave                                   /*Number of processes to terminate*/,
   int resize_allocation                              /*1: Return free resources, 0: Keep resources*/,
   MPI_Errhandler errhandler                          /*Error handler to use in session init*/
);
```

Callback function to prepare session finalize:

```
int (MPIX_Session_reinit_preparation_function)
    (MPI_Session session, int rank, int size,
     int leave, void *userdata);
```

Callback function to resume after session re-init:

```
int (MPIX_Session_reinit_resume_function)
    (MPI_Session session, int old_rank, int old_size,
     int new_rank, int new_size, void *userdata);
```

Example #1: Shrink MPI application

Re-init for terminating ranks;
This call will not return here

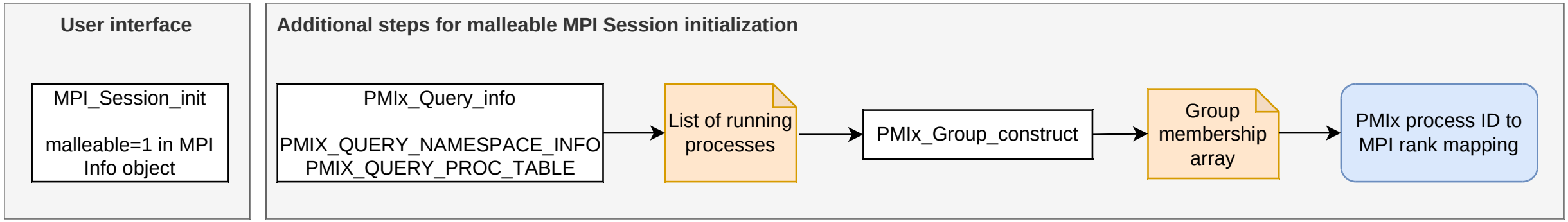
Re-init for staying processes

Shrink complete;
Continue with less processes

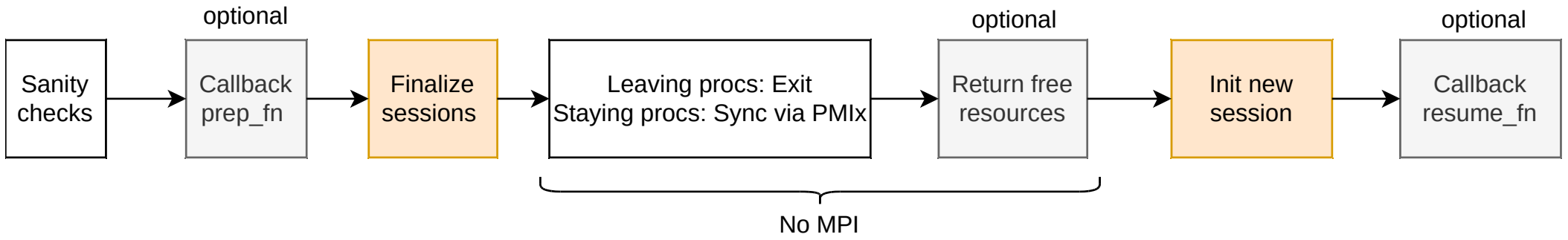
```c
int main(int argc, char *argv[])
{
    […] /* Init variables */
    MPI_Info_set(sinfo, "malleable", "1"); /* Set malleable parameter in info object */
    MPI_Session_init(sinfo, MPI_ERRORS_ARE_FATAL, &session); /* Init session */
    […] /* Create MPI Objects and do work...*/
    sessions_for_reinit[0] = &session;
    […] /* Clean-up old MPI objects as preparation for shrink */
    if (rank == 1 || rank == 2) {
        MPIX_Session_reinit(1, sessions_for_reinit, &session_NEW, MPI_INFO_NULL,
                            MPIX_SESSION_REINIT_PREP_FN_NULL, NULL,
                            MPIX_SESSION_REINIT_RESUME_FN_NULL, NULL,
                            /* This rank terminates */ 1,
                            /* Total of 2 terminating ranks */ 2,
                            /* Keep resources */ 0,
                            MPI_ERRORS_ARE_FATAL);
    } else {
        MPIX_Session_reinit(1, sessions_for_reinit, &session_NEW, MPI_INFO_NULL,
                            MPIX_SESSION_REINIT_PREP_FN_NULL, NULL,
                            MPIX_SESSION_REINIT_RESUME_FN_NULL, NULL,
                            /* This rank stays */ 0,
                            /* Total of 2 terminating ranks */ 2,
                            /* Keep resources */ 0,
                            MPI_ERRORS_ARE_FATAL);
    }
    […] /* Create new MPI objects and continue with work, cleanup at the end */
    MPI_Session_finalize(&session_NEW);
}
```

`MPIX_Session_reinit`: Implementation details

"Malleable" MPI Sessions

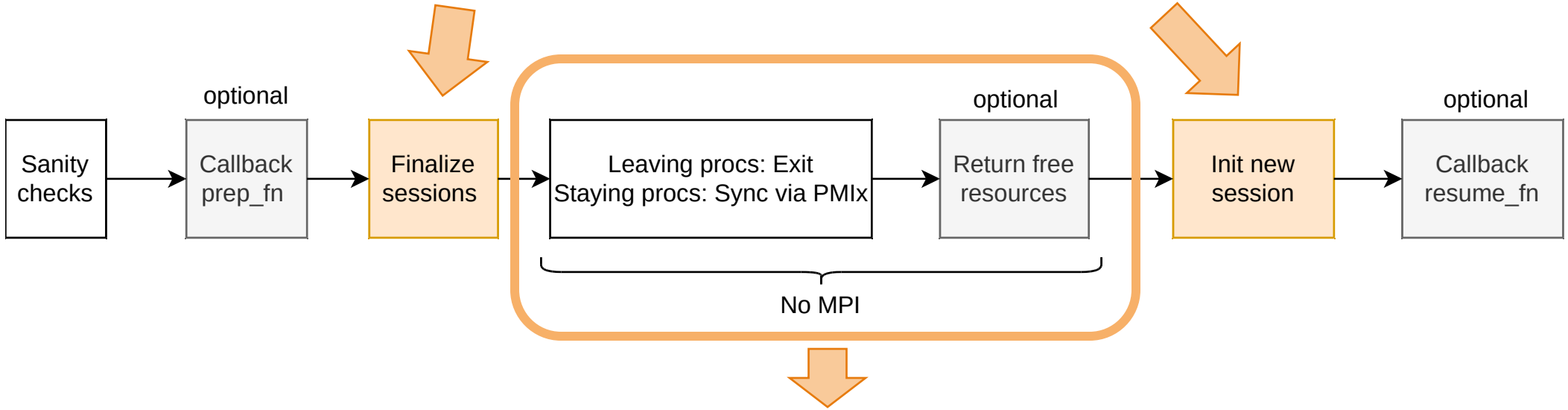


Steps of MPI Session re-initialization



Why add a new MPI interface for session re-initialization?

Couldn't we simply use `MPI_Session_finalize` and `MPI_Session_init`?

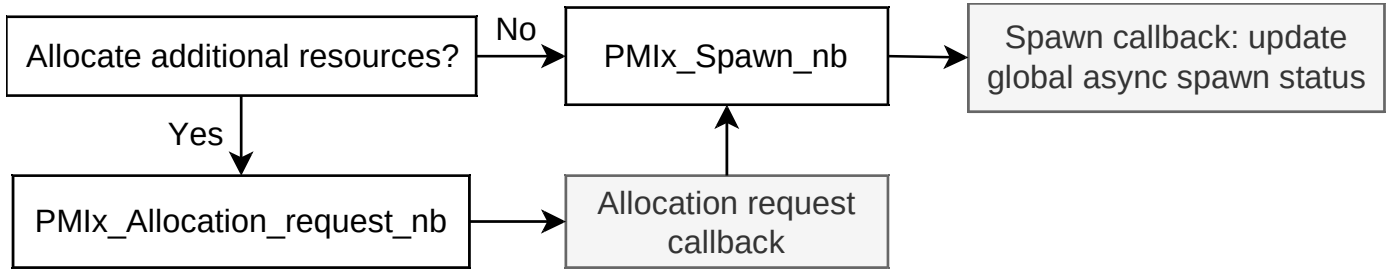


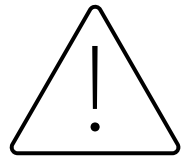
- These steps are important for a **well-defined** resource state before initialization of the new session
- There is no **easy-to-use** user interface to do these steps outside of the MPI library
- Give **optimization potentials** to implementors, e.g., for connection teardown and (re-)creation

Proposed MPI eXtensions for asynchronous resource allocation and spawning

Trigger asynchronous resource allocation and spawning (called in all processes)

```
int MPIX_Spawn_async(
   const char *command   /*see MPI_Comm_spawn*/,
   char *argv[]          /*see MPI_Comm_spawn*/,
   int maxprocs          /*see MPI_Comm_spawn*/,
   MPI_Info info         /*see MPI_Comm_spawn*/,
   int root              /*see MPI_Comm_spawn*/,
   int resize_allocation /*1: Allocate new resources, 0: Use available resources*/,
   MPI_Info *status      /*Status of spawn operation - output*/
);
```





Only **one** asynchronous spawn operation supported at a time

Obtain status

```
int MPIX_Spawn_status(MPI_Info *status);
```

- Async. spawn available?
- Process spawned?
- Status of spawn operation
- Parameters of ongoing spawn

Example #2: Expand MPI application

Trigger async spawn

Know that new processes are available

Re-init complete, new processes are included

```c
int main(int argc, char *argv[])
{
    […] /* Init variables */
    MPI_Info_set(sinfo, "malleable", "1"); /* Set malleable parameter in info object */
    MPI_Session_init(sinfo, MPI_ERRORS_ARE_FATAL, &session); /* Init session */
    […] /* Create MPI Objects and do work...*/
    if (!spawned) {
       /* Trigger the spawn */
       MPIX_Spawn_async((char *) "./my_app", MPI_ARGV_NULL, /* spawn 2 processes */ 2, MPI_INFO_NULL,
                        /* rank 0 is the root */ 0, /* no additional resources */ 0, &spawn_status);
       /* Wait for spawned processes to become ready */
       MPI_Info_get(spawn_status, "spawn_x_status", MPI_MAX_INFO_VAL, status, &found);
       while (strncmp(status, "complete", MPI_MAX_INFO_VAL) != 0) {
              MPIX_Spawn_status(&spawn_status);
              MPI_Info_get(spawn_status, "spawn_x_status", MPI_MAX_INFO_VAL, status, &found);
       }
       […] /* Clean-up old MPI objects */
       /* Re-init */
       sessions_for_reinit[0] = &session;
       MPIX_Session_reinit(1, sessions_for_reinit, &session_NEW, MPI_INFO_NULL,
                           MPIX_SESSION_REINIT_PREP_FN_NULL, NULL,
                           MPIX_SESSION_REINIT_RESUME_FN_NULL, NULL, 0, 0, 0, MPI_ERRORS_ARE_FATAL);
       […] /* Create new MPI objects and continue with work, cleanup at the end */
       MPI_Session_finalize(&session_NEW);
    } else {
       […] /* clean up */
       MPI_Session_finalize(&session); /* Spawned process, finalize session */
    }
}
```

Active and passive malleability with MPI

Active malleability

- Application decides to change number of ranks, e.g., based on computational phases
- Application triggers the procedure using proposed MPI eXtensions

Passive malleability

- Resource changes requested/ proposed from external sources (e.g., scheduler, monitoring system)
- Requests/ proposals relayed to application, e.g., via `MPIX_Spawn_status` or process sets
- Application evaluates requests/ proposals
- Application triggers procedure using proposed MPI eXtensions

Status

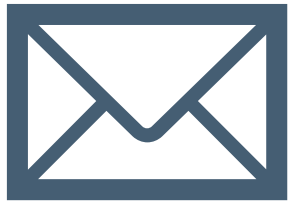
- Code review ongoing @ ParTec
  - Current version publicly accessible via DEEP-SEA's software release
  - https://gitlab.jsc.fz-juelich.de/deep-sea/wp5/software/psmpi/-/tree/deep-sea

- Testing on semi-production system at Jülich Supercomputing Centre, Germany
  - Requires process manager with sufficient PMIx support
    - Ongoing work to enhance PMIx support in ParTec's process manager ParaStation Management
    - Use PRRTE process manager for now
  - Interaction with scheduler: Static resource allocations
  - Results
    - Expansion successful on single and multiple nodes
    - Shrink successful on single node, multiple node case needs changes in PRRTE

MPI Forum

- Thank you!
- Your questions and feedback are highly appreciated!
- Is there an interest to bring this into the MPI standard?

Contact



Sonja Happ

sonja.happ@par-tec.com

www.par-tec.com

ParTec AG

Munich, Germany

Modified Example #1: Shrink MPI application with callbacks

```c
struct foo {...}; /* User data structure */
```

```c
int cleanup(MPI_Session session, int rank,
            int size, int leave, void *userdata)
{
    /* Cleanup MPI objects for session
       Redistribute data if required */
    struct foo* myfoo = (struct foo*) userdata;
    MPI_Group_free(...);
    MPI_Comm_free(...);
    [...]
}
```

```c
int resume(MPI_Session session, int old_rank,
           int old_size, int new_rank,
           int new_size, void *userdata)
{
    /* Create MPI objects for new session
       Redistribute data if required */
    struct foo* myfoo = (struct foo*) userdata;
    MPI_Group_from_session_pset(...);
    MPI_Comm_create_from_group(...);
    [...]
}
```

```c
int main(int argc, char *argv[])
{
    […] /* Init variables */
    struct foo myfoo; /* User data instance */
    MPI_Info_set(sinfo, "malleable", "1"); /* Set malleable parameter in info object */
    MPI_Session_init(sinfo, MPI_ERRORS_ARE_FATAL, &session); /* Init session */
    […] /* Create MPI Objects and do work...*/
    sessions_for_reinit[0] = &session;
    if (rank == 1 || rank == 2) {
        MPIX_Session_reinit(1, sessions_for_reinit, &session_NEW, MPI_INFO_NULL,
                            &cleanup, (void *) &myfoo,
                            &resume, (void *) &myfoo,
                            /* This rank terminates */ 1,
                            /* Total of 2 terminating ranks */ 2,
                            /* Keep resources */ 0,
                            MPI_ERRORS_ARE_FATAL);
    } else {
        MPIX_Session_reinit(1, sessions_for_reinit, &session_NEW, MPI_INFO_NULL,
                            &cleanup, (void *) &myfoo,
                            &resume, (void *) &myfoo,
                            /* This rank stays */ 0,
                            /* Total of 2 terminating ranks */ 2,
                            /* Keep resources */ 0,
                            MPI_ERRORS_ARE_FATAL);
    }
    […] /* Continue with work, cleanup at the end */
    MPI_Session_finalize(&session_NEW);
}
```